

Rapport TP

Kata : Troupe théâtrale Génie Logiciel Avancé

Nom : HADID Prénom : Amine Groupe 3

Introduction

Le présent rapport porte sur un travail pratique qui vise à améliorer un système de gestion de facturation pour des représentations théâtrales. Ce TP offre une opportunité précieuse d'acquérir des compétences en matière d'optimisation de code, de développement de nouvelles fonctionnalités et de préparation à des évolutions futures. L'exercice s'inscrit dans un contexte où le code existant nécessite des améliorations, notamment en ce qui concerne l'efficacité, la maintenabilité, et la flexibilité.

Au cours de ce TP, nous aborderons les différentes étapes, de l'installation de l'environnement de développement à l'ajout de fonctionnalités et à la gestion des informations clients. L'objectif ultime est d'offrir un système robuste capable de générer des factures pour des pièces de théâtre, tout en prenant en compte les besoins actuels et futurs du client. Ce rapport détaillera chaque phase du processus, mettant en lumière les décisions prises et les améliorations apportées au code existant.

Partie 0 Installation

Pour commencer ce travail pratique, j'ai téléchargé les fichiers du TP à partir de la source fournie en annexe. L'accès aux ressources et liens utiles a été essentiel pour obtenir le matériel nécessaire. Il est important de noter que ce TP nécessite que JDK 17 soit installé sur l'ordinateur. Pour les utilisateurs qui ne disposent pas de Java Development Toolkit (JDK) 17, il est recommandé de le télécharger depuis le site d'Adoptium. Il est à noter que ce TP n'est pas compatible avec Java 19, car il n'y a pas encore de version de Gradle compatible pour ce JDK. Par conséquent, si un utilisateur a Java 19 installé, il est nécessaire de le désinstaller et de revenir à Java 17, qui est la dernière version LTS disponible.

L'utilisation de Git pour la gestion de version et Gradle est également recommandée, étant donné que ces outils sont utilisés dans ce TP, tout comme dans le TP1.

Partie 1 Nettoyage du code

Cette partie du TP consistait à effectuer des améliorations sur le code existant, en mettant l'accent sur le fichier StatementPrinter. Plusieurs suggestions ont été prises en compte pour améliorer la qualité du code, notamment :

- L'utilisation d'un StringBuffer (ou StringBuilder) au lieu de concaténer des chaînes de caractères en boucle, afin d'améliorer les performances.

```
StringBuilder result = new StringBuilder();
```

- L'amélioration de la lisibilité du code en ajoutant des commentaires, des noms de variables explicites, etc.

```
//quelque exemples des commentaires  
//get the solde points of the custome  
// print line for this order
```

- La création de variables statiques pour les types de pièces de théâtre, ce qui a permis d'ajouter un contrôle sur le type de pièce à la création de l'objet.

```
public static final String TRAGEDY = "tragedy";  
public static final String COMEDY = "comedy";
```

- La gestion directe des sommes en flottant au lieu de travailler en entier / 100, ce qui a amélioré la précision des calculs.

```
Double thisAmount = 0.0;// un double pour les montants
```

En outre, la possibilité d'ajouter des tests unitaires a été envisagée pour sécuriser les parties du code qui ne sont pas couvertes par les tests d'acceptation existants. Il est important de noter que nous avons eu la liberté de modifier tout le code du TP pour atteindre ces objectifs.

La methode print ou toTEXT

```
import java.text.NumberFormat;  
import java.util.*;  
  
public String print(Invoice invoice, HashMap<String, Play> plays, Customer  
customer) {  
    int totalAmount = 0;  
    int volumeCredits = 0;
```

```

StringBuilder result = new StringBuilder();
int cus=customer.soldepoints; //get the solde points of the customer
result.append(String.format("Statement for %s\n", invoice.customer.name));
result.append(String.format("Your points %d \n", cus));

NumberFormat frmt = NumberFormat.getCurrencyInstance(Locale.US);

for (Performance perf : invoice.performances) {
    Play play = plays.get(perf.playID);

    CalculClass calcul = new CalculClass();
    double thisAmount = calcul.calcul(perf, play,customer);

    // add volume credits
    volumeCredits += Math.max(perf.audience - 30, 0);
    // add extra credit for every ten comedy attendees
    if (Play.playType.COMEDY.equals(play.type)) volumeCredits +=
Math.floor(perf.audience / 5);

    // print line for this order
    result.append(String.format("  %s: %s (%s seats)\n", play.name,
frmt.format(thisAmount), perf.audience));
    totalAmount += thisAmount;

}
if (cus>150){
    cus=cus-150;
    totalAmount-=15;
}
cus=cus+volumeCredits;
//customer.setsoldepoints(cus); //set the solde points of the customer
result.append(String.format("Amount owed is : %s\n",
frmt.format(totalAmount)));
result.append(String.format("You earned : %s points\n", volumeCredits));
result.append(String.format("Points left : %s points\n", cus));
return result.toString();
}

```

Partie 2 Ajout renduHTML

Une nouvelle exigence a été présentée dans cette partie du TP, à savoir l'ajout d'un rendu HTML au système existant. Pour ce faire, j'ai suivi les instructions suivantes :

- Une nouvelle classe a été créée pour gérer le calcul de la facture et stocker l'information dans un objet. Cela a nécessité des modifications dans la classe Invoice, et le choix de créer une nouvelle classe était également une option.

```

public class CalculClass {
    static String TRAGEDY = "tragedy";
    static String COMEDY = "comedy";

    public double calcul(Performance perf, Play play, Customer cus){
        Double thisAmount = 0.0; // un double pour les montants

        switch (play.type) {
            case TRAGEDY:
                thisAmount = 400.00; // 400.0 au lieu de 40000
                if (perf.audience > 30) {
                    thisAmount += 10 * (perf.audience - 30); // 10 au lieu de 1000
                }
                break;
            case COMEDY:
                thisAmount = 300.00; // 300.0 au lieu de 30000
                if (perf.audience > 20) {
                    thisAmount += 100 + 5.0 * (perf.audience - 20); // 100 au lieu de
10000 et 5.0 au lieu de 500
                }
                thisAmount += 3.00 * perf.audience; // 3.0 au lieu de 300
                break;
        }

        return thisAmount;
    }
}

```

- Une méthode a été ajoutée : toHTML, qui génère la facture au format HTML. Cette approche permet de maintenir l'affichage "texte" tout en proposant une nouvelle option HTML. Il est important de noter que la conformité avec les normes du W3C a été respectée lors de la création du format HTML.

```

public String toHTML(Invoice invoice, HashMap<String, Play> plays, Customer
customer) {

    int totalAmount = 0;
    int volumeCredits = 0;
    StringBuilder result = new StringBuilder();
    NumberFormat frmt = NumberFormat.getCurrencyInstance(Locale.US);
    int cus=customer.soldepoints; //get the solde points of the customer

    for (Performance perf : invoice.performances) {
        Play play = plays.get(perf.playID);
    }
}

```

```

        CalculClass calcul = new CalculClass();

        double thisAmount = calcul.calcul(perf, play, customer);
        // add volume credits
        volumeCredits += Math.max(perf.audience - 30, 0);
        // add extra credit for every ten comedy attendees
        if (Play.playType.COMEDY.equals(play.type)) volumeCredits +=
Math.floor(perf.audience / 5);

        // print line for this order
        //result.append(String.format(" %s: %s (%s seats)\n", play.name,
frmt.format(thisAmount), perf.audience));
        totalAmount += thisAmount;

    }
    result.append(String.format("<p><strong>Invoice</strong> </p>\n ") );
    result.append(String.format("<p><strong>Client :</strong> %s </p>\n",
invoice.customer.name));

    if (cus>150){
        cus=cus-150;
        totalAmount-=15;
    }
    cus=cus+volumeCredits;
    //customer.setsoldepoints(cus); //set the solde points of the customer

    // Génération du détail de la facture au format HTML
    result.append("<style>");
    result.append("table { border-collapse: collapse; width: 50%; }");
    result.append("th, td { border: 1px solid black; padding: 8px; text-
align: left; }");
    result.append("</style>");
    result.append("<table>\n");
    result.append("<tr><th>Play</th><th>Seats</th><th>Amount</th></tr>\n");
    double thisAmount = 0;
    for (Performance perf : invoice.performances) {
        Play play = plays.get(perf.playID);
        CalculClass calcul = new CalculClass();
        thisAmount = calcul.calcul(perf, play, customer);
        result.append(String.format(" <tr><td>%s</td><td>%s</td><td>%s</td></
tr>\n", play.name, perf.audience, frmt.format(thisAmount)));
    }
    result.append(String.format("<tr><td><strong> Amount owed
is</strong></td><td>%s</td></tr>\n", frmt.format(totalAmount)));
    result.append(String.format("<tr><td><strong>You
earned</strong></td><td> %d points</td></tr>\n", volumeCredits));

```

```

        result.append(String.format("<tr><td><strong>Points  
left</strong></td><td> %d points</td></tr>\n", cus));
        result.append("</table>\n");

        result.append(" Payment is required under 30 days, we can brake your knees  
if you dont do so\n ");
        result.append("</body></html>\n");

        return result.toString();
    }

```

- Un test d'acceptation pour le format HTML a également été mis en place pour garantir la qualité du rendu.

```

import static org.approvaltest.Approvals.verifyHtml
void exampleStatementtohtml() {

    HashMap<String, Play> plays = new HashMap<>();
    plays.put("hamlet", new Play("Hamlet", Play.playType.TRAGEDY));
    plays.put("as-like", new Play("As You Like  
It", Play.playType.COMEDY));
    plays.put("othello", new Play("Othello", Play.playType.TRAGEDY));

    Invoice invoice = new Invoice(customer.name, List.of(
        new Performance("hamlet", 55),
        new Performance("as-like", 35),
        new Performance("othello", 40)));

    StatementPrinter statementPrinter = new StatementPrinter();
    var result = statementPrinter.toHTML(invoice, plays, customer);

    verifyHtml(result);
}

```

Partie 3 Se simplifier la vie pour le futur

Cette partie du TP a pris en compte la possibilité d'ajouter de nouveaux types de représentation dans le futur.

- L'ajout d'une énumération dans Play.java pour représenter les différents types de représentation.

```

public enum playType {
    TRAGEDY, COMEDY }

```

L'objectif ici est de rendre le système extensible et facile à maintenir pour de futures évolutions.

Partie 4 Gestion des clients

Dans cette partie, une nouvelle fonctionnalité a été introduite pour gérer les informations des clients. La classe Customer a été créée pour stocker les informations suivantes pour chaque client :

- Nom du client.
- Numéro client.
- Solde de points de fidélité.

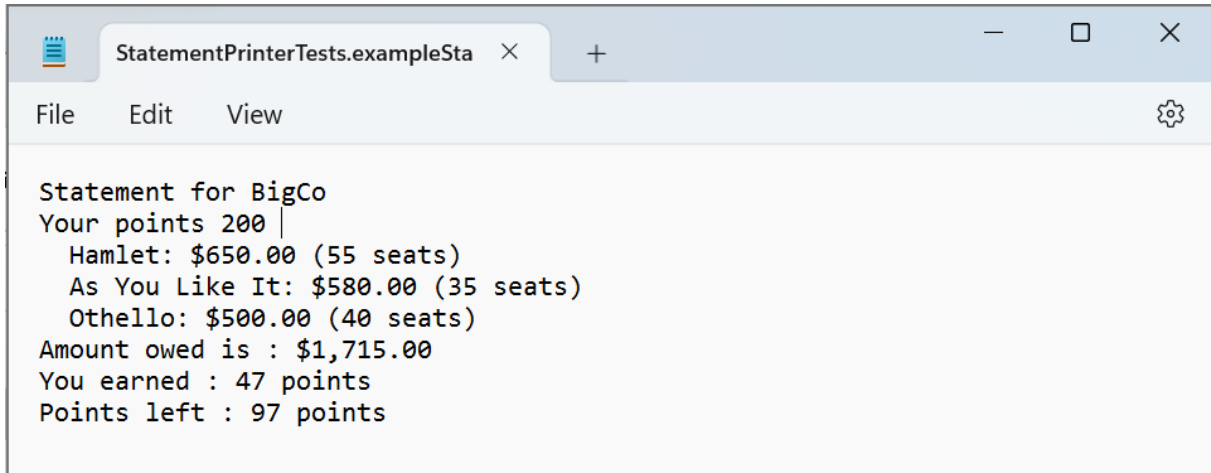
```
public class Customer {
    String name;
    int numcust;
    int soldepoints;
    public Customer(String name, int numcust, int soldepoints) {
        this.name = name;
        this.numcust = numcust;
        this.soldepoints = soldepoints; }
    public String getname(){
        return name;
    }
    public int getnumcust(){
        return numcust;
    }
    public int getsoldepoints(){
        return soldepoints;
    }
    public void setsoldepoints(int soldepoints){
        this.soldepoints=soldepoints;
    } }
}
```

Une règle spécifique a été mise en place dans la classe de calcul : si un client a plus de 150 points sur son solde de fidélité lors de l'émission d'une facture, une réduction de 15€ est appliquée, et 150 points sont déduits de son solde de fidélité.

```
if(cus.soldepoints>150){
    thisAmount-=15;
    cus.soldepoints-=150;
}
```

Les informations relatives aux clients ont été ajoutées aux affichages textuels et HTML, et les tests ont été mis à jour en conséquence.

```
Customer customer = new Customer("BigCo", 1522, 200);
```



```
Statement for BigCo
Your points 200
  Hamlet: $650.00 (55 seats)
  As You Like It: $580.00 (35 seats)
  Othello: $500.00 (40 seats)
Amount owed is : $1,715.00
You earned : 47 points
Points left : 97 points
```

Invoice

Client : BigCo

Play	Seats	Amount
Hamlet	55	\$650.00
As You Like It	35	\$580.00
Othello	40	\$500.00
Amount owed is	\$1,715.00	
You earned	47 points	
Points left	97 points	

Payment is required under 30 days, we can brake your knees if you dont do so

Remarque : on a deux fichier approved.html et approved.txt qui contiennent les impressions de toHTML et toTEXT et donc à chaque fois on change les infos dans les tests on reçoit les fichiers received.html et received.txt qui vont être comparer aux deux fichiers mentionnées en premier et le test ne marchera, donc il faut lancer la commande move-item pour faire un overwrite et copier les fichiers.received sur les fichier.approved

Move-Item -Path

"C:\Users\LENOVO\OneDrive\Bureau\TP2GL\base\.\src\test\java\StatementPrinterTests.exampleStatementPrinttotext.received.txt" -Destination

"C:\Users\LENOVO\OneDrive\Bureau\TP2GL\base\.\src\test\java\StatementPrinterTests.exampleStatementPrinttotext.approved.txt" -Force

Move-Item -Path

"C:\Users\LENOVO\OneDrive\Bureau\TP2GL\base\.\src\test\java\StatementPrinterTests.exampleStatementtohtml.received.html" -Destination

"C:\Users\LENOVO\OneDrive\Bureau\TP2GL\base\.\src\test\java\StatementPrinterTests.exampleStatementtohtml.approved.html" -Force

Conclusion

Ce travail pratique a été l'occasion d'explorer différentes améliorations du code, d'ajouter un rendu HTML au système existant, de se préparer à de futures extensions et d'introduire la gestion des informations clients. Ces améliorations ont permis d'augmenter la qualité et la flexibilité du système, tout en maintenant la conformité avec les exigences initiales du TP.